EXPERIMENT - 11

Transactions & Concurrency Control in MySQL

Part A: Prevent Duplicate Enrollments Using Locking

Simulate concurrent users attempting to enroll students in courses. Implement a mechanism that prevents two users from enrolling the same student into the same course simultaneously by using transactions and unique constraints.

Table: StudentEnrollments

Columns:
- enrollment_id (INT, Primary Key)
- student_name (VARCHAR(100))
- course_id (VARCHAR(10))
- enrollment_date (DATE)

Constraints:
Each student can enroll in a course only once.
The pair (student_name, course_id) must be unique.

SQL Script

```
CREATE TABLE StudentEnrollments (
  enrollment_id INT NOT NULL AUTO_INCREMENT,
  student_name VARCHAR(100) NOT NULL,
  course_id VARCHAR(10) NOT NULL,
  enrollment_date DATE NOT NULL,
  PRIMARY KEY (enrollment_id),
  UNIQUE KEY uq_student_course (student_name, course_id)
) ENGINE=InnoDB;

INSERT INTO StudentEnrollments (student_name, course_id, enrollment_date)
VALUES
  ('Ashish', 'CSE101', '2024-07-01'),
  ('Smaran', 'CSE102', '2024-07-01'),
 ('Vaibhav','CSE101', '2024-07-01');
```

Expected Output:

If two users try to enroll 'Ashish' in 'CSE101', only the first will succeed; the second will get a constraint violation.

Part B: Use SELECT FOR UPDATE to Lock Student Record

Use row-level locking via SELECT FOR UPDATE to prevent conflicts. Simulate a situation where a student is being verified before enrollment and locked until confirmation, preventing other users from updating it simultaneously.

SQL Script (Example):

```
START TRANSACTION;
SELECT * FROM Students WHERE student_name = 'Ashish' FOR UPDATE;
-- Perform verification
INSERT INTO StudentEnrollments (student_name, course_id, enrollment_date)
VALUES ('Ashish', 'CSE101', '2024-07-02');
COMMIT;
```

Expected Output:
User B will be blocked until User A finishes the transaction.

Part C: Demonstrate Locking Preserving Consistency

Demonstrate how locking preserves data consistency when multiple users attempt concurrent updates. Show how update conflicts are avoided when row-level locks are used appropriately in transactions.

SQL Script (Example):

```
Session A:
START TRANSACTION;
SELECT * FROM StudentEnrollments WHERE student_name='Ashish' AND
course_id='CSE101' FOR UPDATE;
UPDATE StudentEnrollments SET enrollment_date = '2024-07-10' WHERE
student_name='Ashish' AND course_id='CSE101';
COMMIT;
```

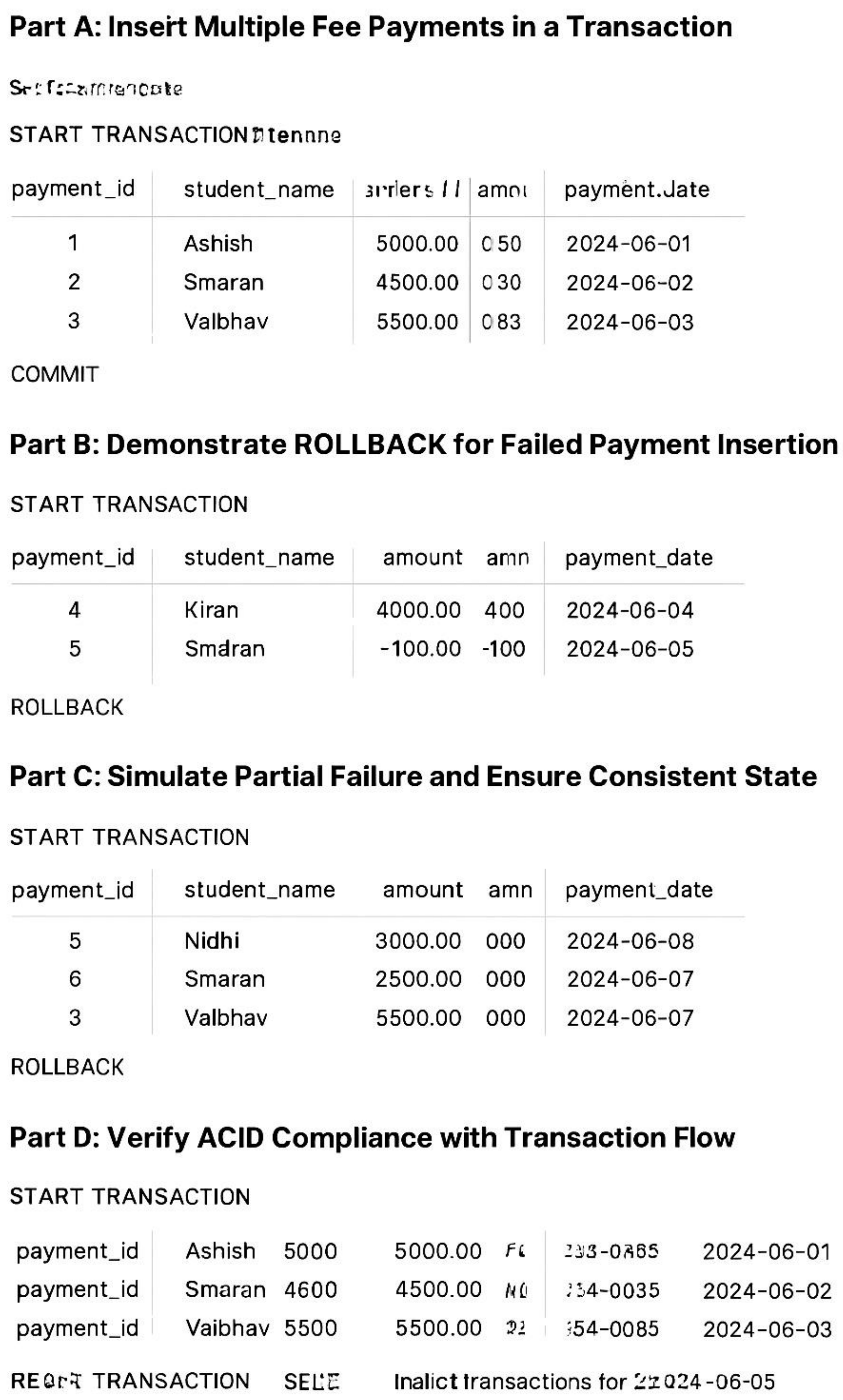
Session B:

```
START TRANSACTION;
SELECT * FROM StudentEnrollments WHERE student_name='Ashish' AND
course_id='CSE101' FOR UPDATE;
UPDATE StudentEnrollments SET enrollment_date = '2024-07-20' WHERE
student_name='Ashish' AND course_id='CSE101';
COMMIT;
```

Expected Output:

Operations are serialized. Final state is consistent, reflecting last committed update.

Sample Output Image



Part A: Insert Multiple Fee Payments in a Transaction

Sr:f::amrenoote

START TRANSACTION tennne

payment_id	student_name	arrlers / /	amou	payment.Jate
1	Ashish	5000.00	0 50	2024-06-01
2	Smaran	4500.00	0 30	2024-06-02
3	Valbhav	5500.00	0 83	2024-06-03

COMMIT

Part B: Demonstrate ROLLBACK for Failed Payment Insertion

START TRANSACTION

payment_id	student_name	amount	amn	payment_date
4	Kiran	4000.00	400	2024-06-04
5	Smdran	-100.00	-100	2024-06-05

ROLLBACK

Part C: Simulate Partial Failure and Ensure Consistent State

START TRANSACTION

payment_id	student_name	amount	amn	payment_date
5	Nidhi	3000.00	000	2024-06-08
6	Smaran	2500.00	000	2024-06-07
3	Valbhav	5500.00	000	2024-06-07

ROLLBACK

Part D: Verify ACID Compliance with Transaction Flow

START TRANSACTION

payment_id	Ashish	5000	5000.00	FC	233-0865	2024-06-01
payment_id	Smaran	4600	4500.00	NC	234-0035	2024-06-02
payment_id	Vaibhav	5500	5500.00	22	354-0085	2024-06-03

REQrT TRANSACTION SELE Inalict transactions for 22024-06-05